CS2042 - Unix Tools

Fall 2010

Lecture 5

Hussam Abu-Libdeh

based on slides by David Slater

September 17, 2010

Reasons to use Unix

Reason #42 to use Unix: Wizardry

Mastery of Unix makes you a wizard

- need proof?
- here is the profile page of the employees of a real estate company
  - http://www.deansproperty.com.au/Home/Profiles
- can you spot the IT guy?

sort

Sorts the lines of a text file alphabetically.

- `sort -r u file`
  - sorts the file in reverse order and deletes duplicate lines.
- `sort -n -k 2 -t : file`
  - sorts the file numerically by using the second column, separated by a colon

Consider a file (`numbers.txt`) with the numbers 1, 5, 8, 11, 62 each on a separate line, then:

```
$ sort numbers.txt
1
11
5
62
8
```

```
$ sort numbers.txt -n
1
5
8
11
62
```

`uniq`

- `uniq file` - Discards all but one of successive identical lines
- `uniq -c file` - Prints the number of successive identical lines next to each line

Search and Replace

The Translate Command

`tr [options] <set1> [set2]`

- Translate or delete characters
- Sets are strings of characters
- By default, searches for strings matching set1 and replaces them with set2

Example:

`echo somefile | tr 'AEIOU' 'aeiou'` - changes all capital vowels to lower case vowels

Redirection revisited

- `tr` only receives input from *standard input* (stdin)
  - i.e. keyboard input
- What if we want to operate on files?
  1. Piping: `cat somefile | tr 'AEIOU' 'aeiou'`
  2. Input redirection: `tr 'AEIOU' 'aeiou' < somefile`

Redirection revisited

Input/Output Streams

- #0 : Standard input stream; STDIN (usually keyboard but can be redirected)
  - to redirect standard input, use the `<` operator
  - `command < file`
- #1 : Standard output stream; STDOUT (usually terminal console but can be redirected)
  - to redirect standard output, use the `>` operator
  - `command > file`
- #2 : Standard error stream; STDERR (depends on system setting, can also be redirected)
  - to redirect standard error, use the `>` operator and specify the stream by number (2)
  - `command 2> file`

Can combine two streams together by using `2>&1`
This says: send standard error to where standard output is going.
Useful for debugging/catching error messages.

Redirection revisited

Bash processes I/O redirection from left to right, allowing us to do fun things like this:

Example:

Let's delete everything but the numbers from test1.txt, then store them in test2.txt

- `tr -cd '0-9' < test1.txt > test2.txt`

Some Simple Examples

Example:

`echo *` prints everything in the directory, separated by spaces. Let's separate them by newlines instead:

- `echo * | tr ' ' '\n'` – replaces all spaces with newlines

Example:

Let's print a file in all uppercase:

- `tr 'a-z' 'A-Z' < test.txt` - prints the contents of text.txt in all caps

tr and sets

- If the two sets passed to tr are the same length then the first character in the first set is replaced by the first in the second and so on.
- If the second is shorter than the first, then they are matched and all extra are changed to the last character
- If the first is shorter, then only those corresponding characters in the second matter

Example:

```
echo "abcdefghijklmnopqrstuvwxyz" | tr 'a-z' 'a'
aaaaaaaaaaaaaaaaaaaaaaaaaa

echo "abcdefghjiklmnopqrstuvwxyz" | tr 'a-z' 'wxyz'
wxyzzzzzzzzzzzzzzzzzzzzzzz
```

tr options

tr has some very useful options:

tr options

- `tr -d <set>` - delete all characters that are in the set
- `tr -c <set1> [set2]` - complements set1 before replacing it with set2

Example:

Lets print a file with all non-letters removed:

- `tr -cd 'a-zA-Z' < somefile` - removes non-letters from somefile

tr and character ranges

tr does not understand regular expressions (and really for the task it is designed for they don't make sense), but it **does** understand characte ranges and POSIX character sets such as `[:alpha:]`

Reminder:

- `[:alnum:]` - alphanumeric characters
- `[:alpha:]` - alphabetic characters
- `[:digit:]` - digits
- `[:punct:]` - punctuation characters
- `[:lower:]` - lowercase letters
- `[:upper:]` - uppercase letters
- `[:space:]` - whitespace characters

Substitution Cypher

We can use `tr` to do a simple substitution cypher. Create a text file called "cypher" with some reordering of the alphabet. Then to encode we would just do

- `tr 'a-z' `cat cypher` < file > encodedfile`

and to decode we would do

- `tr `cat cypher` 'a-z' < encodedfile > decodedfile`

Note the use of backticks around `cat cypher`. By doing this the shell expands this command and passes the output to `tr`.

`sed`

sed is a *stream editor*. We will only cover the basics, as it is a completely programming language!

Stream Editor

`sed [options] [script] [file]`

- Straem editor for filtering and transforming text
- We will focus on `sed 's/<regexp>/<text>' [file]`
- This form replaces anything that matches `<regexp>` with `<text>`.
- `sed` goes line by line searching for the regular expression.

What is the difference between `sed` and `tr`?

- `sed` can match regular expressions!
- `sed` also does lots of other stuff

Basic Example:

Example:

```
sed 's/not guilty/guilty/g' filename
```

Replaces not guilty with guilty everywhere in the file

What happens if we don't have the g?

Basic Example:

Example:

```
sed 's/not guilty/guilty/g' filename
```

Replaces not guilty with guilty everywhere in the file

What happens if we don't have the g?

Without the g, it will only do one substitution per line.

sed deletion

Just like with `tr` we can do deletion with `sed`

sed deletion

- `sed '/regexp/d'` - deletes all lines that contain regexp

Example

```
sed '/[Dd]avid/d' filename > filename2
```

- deletes all **lines** that contain either David or david and saves the file as `filename2`.

sed understands regular expressions!

The power of sed is that it treats everything between the first pair of /'s as a regular expression. So we could do

```
sed 's/[[:alpha:]]\{1,3\}[[:digits:]]*@cornell\.edu/cornell email
removed/g' file
```

to print a file with all cornell email addresses removed.

use -r to use extended regular expressions.

sed is greedy!

sed matches a given regular expression to the **longeset** string as soon as possible:

```
$ echo filename1
a b col d e f lapse h i j k lapse m n
$ sed 's/col.*lapse/collapse/g' filename1
```

sed is greedy!

sed matches a given regular expression to the **longeset** string as soon as possible:

```
$ echo filename1
a b col d e f lapse h i j k lapse m n
$ sed 's/col.*lapse/collapse/g' filename1
a b collapse m n

$ echo filename2
a b c col d e f col g h i lapse
sed 's/col.*lapse/collapse/g' filename2
```

sed is greedy!

sed matches a given regular expression to the **longeset** string as soon as possible:

```
$ echo filename1
a b col d e f lapse h i j k lapse m n
$ sed 's/col.*lapse/collapse/g' filename1
a b collapse m n

$ echo filename2
a b c col d e f col g h i lapse
sed 's/col.*lapse/collapse/g' filename2
a b c collapse
```

sed can save the string

Another Example:

```
sed 's/^\([A-Z][A-Za-z]*\), \([A-Z][A-Za-z]*\)/\2 \1/' filename
```

- Searches for an expression at the beginning of the line of the form `e1, e2` where `e1` and `e2` are "words" starting with capital letters.
- Placing an expression inside ( ) tells the editor to save whatever string matches the expression
- Since ( ) are special characters we escape them; i.e. by using `\( \)`
- We access the saved strings as `\1, \2`.
- This script for example could convert a database file from

`Lastname, Firstname` to `Firstname Lastname`

More sed

You can specify which lines to check by numbers or with regular expressions:

`sed '1,20s/john/John/g' filename` - checks lines 1 to 20

`sed '/^The/s/john/John/g' filename` - checks lines that start with The

& cooresponds to the pattern found:

```
sed  's/[a-z]\+/"&"/g' filename
```

replaces words with words in quotes For more on sed check out

http://www.grymoire.com/Unix/Sed.html

Even more sed

How could we use sed to remove a specific regular expression?

Even more sed

How could we use sed to remove a specific regular expression? `sed 's/regexp/ /g' file`

Example:

```
sed 's/[[:alnum:]]/ /g' Frankenstein.txt
```

Examples:

Let's strip the directory prefix from our pathnames (i.e. convert `/usr/local/src` to `src`

Example:

```
pwd | sed 's/.*//'
```

- Translates anything preceeding (and including) a frontslash to nothing
- Note the backslash-escaped frontslash

sed scripting

sed is a complete programming language and we can write sed scripts.

- Any file that begins with #! is a script file (we will talk more about this next week).

Example

- Create a new text file named trim.sed

```
#! /usr/bin/sed -f
s/^ *//
s/ *$//
```

You can run this script from the shell like any other program:

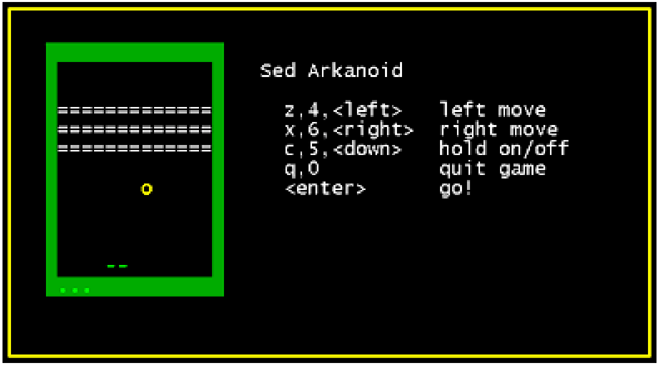
- `echo " this is a test " | ./trim.sed`

```
this is a test
```

We now have a script that trims leading and trailing whitespace!

Sed Arkanoid

Sed is a complete programming language. In fact people have written entire games as sed scripts.



```
Sed Arkanoid

   z,4,<left>     left move
   x,6,<right>    right move
   c,5,<down>     hold on/off
   q,0            quit game
   <enter>        go!
```

http:aurelio.net/soft/sedarkanoid/

Putting things together

Example:

We can put some of these together commands together now to do interesting things.

- `history | sort -k 2 | sed 's/^ *[0-9]\ *//' | uniq -c | sort -n +`

Print out the history of commands with how many times each has been run